sevenstax Embedded HTTP Server

“F-Server”

User's Manual

Revision No.: 3.1

State: Release

Author: sevenstax GmbH

Created:	18/11/02
Changed:	15/11/06
Changed by:	Ralf Schwarzer
Last review:	15/11/06
Publication state:	Public
File name:	webserver_manual_3_1.odt

Content

1 Preface....3
2 Features and Limitations....3
3 What else do I need?....5
4 How it works....6
5 Memory consumption and buffers....8
6 Application callable Interfaces....12
7 Data Structures and Enumerated Types....14
8 How the F-Server accesses files and other resources....23
9 Application defined Callback Procedure....26
10 Creating Resources (HTML Files, Bitmaps, etc.)....31
11 Creating dynamic HTML Forms....32
12 Dynamic Controls Examples....35
13 Authentication....39
14 File upload using HTML....41

1 Preface

Today, most computing platforms (like Windows, Linux, BSD, Solaris etc.) are shipped with integrated web browsers and a TCP/IP protocol suite. This makes them excellent tools to be used for communication between humans and embedded devices.

sevenstax offers a tiny HTTP server which is especially designed to work on embedded systems (small memory footprint) and which is able to generate dynamic HTML forms at runtime (called 'F-Server' in this document).

2 Features and Limitations

- **Stand-alone HTTP server.**
  - Possibly one of the smallest web servers in the world (average: 3K ROM, 1K RAM). RAM/ROM usage is highly customisable and can be sometimes under 1K/2K (see description below in this document).
  - Does not need Rx/Tx buffers. Incoming data is read from a stream and controls an internal state machine.
  - Does not need an operating system but can also be used if you have one.
  - Does not need an Unix-style network interface (sockets).
  - Needs minimal system or hardware support (only one timer/counter required).
  - Non-blocking architecture ensures smooth behaviour when using into MCU main loop or operating system task when running under an multitasking OS.
  - Capable of sending HTML pages, GIF, JPEG images, JAVA applets and many more resource types.
  - Comes with full source code, extensive documentation, additional utilities for MS Windows and samples.
  - Entirely written in ISO-C language which makes it easy to compile with standard C-compilers (Those conforming to ISO/IEC 9899:1999).
  - Independent of bus width and processor architecture.

- **Trouble-free operation with most web browsers.**
  - Tested with Netscape, Mozilla!, Internet Explorer, Opera and Konquerer web browser.
  - Operates with HTTP 1.0 and HTTP 1.1.

- **Integrated support for dynamic HTML forms.**
  - Application can change form controls at runtime.
  - Notification on POST events with automatic URL encoding.

    Accepts 'application/x-www-form-urlencoded'
  - Developers can design form templates with common HTML design tools.

- **Multiple simultaneous connections.**
    - The F-Server can be configured to handle many HTTP requests at the same time (like real big web servers do). This is of particular importance if a page contains pictures and other parts, which must be loaded at the same time to complete the page.

3 What else do I need?

- **An embedded system.**
  - Any data bus width from 8...32 bits.
  - Big or little endian architecture.
  - At least 2K of ROM and 1K of RAM are needed by the web server itself.
  - Some additional ROM space for the HTML pages and/or form templates.
  - Also some RAM/ROM for the application, network device drivers and protocols.
  - A millisecond counter either in hardware or software.

- **Network hardware and corresponding protocols**
  - sevenstax embedded HTTP server does not care about underlying components. Thus, it will work with any kind of network, TCP/IP with either modem/PPP, Ethernet/ARP or proprietary ones.

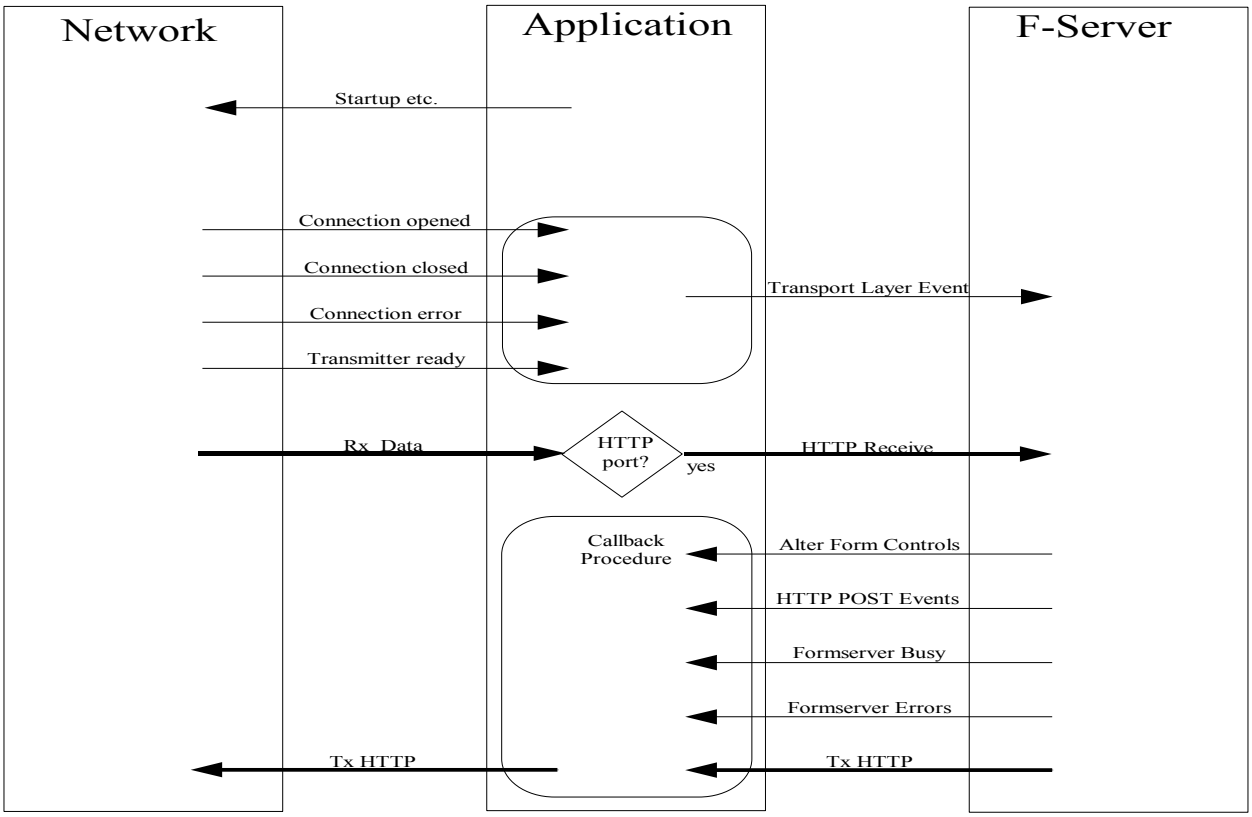
- **Network connection over reliable transport protocol**
  - HTTP naturally sits atop of TCP/IP. Therefore, you should have a TCP stack running on the target system. Of course, sevenstax embedded HTTP server can also be used with other transport layers than TCP/IP.
  - There is no need for a full streaming TCP stack. Because sevenstax HTTP server accepts chunks of data, the underlying TCP does not need to have a FIFO buffer but can deliver received packets to the HTTP server directly.

- **C-compiler and build tools**
  - sevenstax HTTP server compiles trouble-free with today's embedded development tools such as gcc, Keil, Tasking, Renesas, Analog Devices and emC++ 6.

- **Self-made application software which, beside other tasks, ...**
  - ...should call the F-Server's interfaces.
  - ...should receive and act on events from the F-Server.
  - ...should dynamically change a HTTP form's control elements.
  - ...should work with data from HTTP POST commands (variables and values, file upload data).
  - ...should inform the F-Server of network events e.g. when a connection is established or closed.

4 How it works

In the majority of cases the F-Server is driven by a user-made application program which handles both, a network interface and the F-Server itself. This means that the F-Server is a passive component and does always execute on behalf of the application. The following describes a common working relationship of application, network and F-Server.



- **Startup and activation**

  The application initialises the network and F-Server. It optionally enables HTTP authentication and opens a network channel (e.g. TCP port 80) to await a connection.

- **If a connection was established...**

  The application notifies the F-Server of a new connection by calling its interface 'stxHTTP_TransportLayerEvent()'.

- **If data arrives...**

  The application delivers incoming data by calling the F-Server's 'stxHTTP_Receive()' function.

- **If the F-Server has data to send...**

  The F-Server calls an application-defined function to send data off to the network. The application can send the data immediately by calling the network's send function.

- **If the F-Server has got information posted by the a client e.g. web browser...**

  The F-Server calls an application-defined function with all information regarding to the browser's input. The application can decide how to handle the data.

- **If the F-Server wants to create dynamic HTML...**

  The application also receives this in a call to its handler procedure. It now has a chance to modify or completely replace an dynamic HTML element.

  NOTE: In order to prevent system load peaks in HTML pages with many dynamic elements, the F-Server uses two stxHTTP_Tick() to request and generate one dynamic element. So please take care to call stxHTTP_Tick() as often as possible to increase the webserver performance, if many dynamic elements are in use.

- **If the F-Server has completely transmitted a resource....**

  The application receives a notification by a call to its handler-procedure. The application should immediately close the network connection to signal the client the transaction's end.

- **If the network connection ends or was cancelled**

  The application must call the F-Server's stxHTTP_TransportLayerEvent() function in order to bring the F-Server to its idle state.

5 Memory consumption and buffers

As mentioned before, the F-Server is designed to work in low cost embedded devices, which are equipped with limited memory. Therefore, a major design goal was to minimise memory usage. Currently, the F-Server has some buffers which definitions and sizes can be found at 'httpserv.h' and 'features.h'. Those buffer sizes can be changed for some reasons (e.g. optimising RAM usage) but that's not a must. Normally, the default values work well.

Nearly all those definitions below live in 'httpserv.h' and 'features.h'.

The following list shows the configuration of buffer sizes and memory usage:

- **HTTP_MULTI 4 (default)**

  With HTTP_MULTI, you can set up how many instances of the F-Server may exist at the same time. Every instance needs roughly 280 bytes RAM (on an 'x86 system), thus an F-Server configured for 4 instances occupies 1120 bytes of RAM.

  If you need only a single instance (e.g. if you have only straight HTML without bitmaps), set HTTP_MULTI to 0. This will result in approximately 20% of ROM savings because instance management and connection multiplexer will not be added to the executable.

- **HTTP_DYNINFO_ENABLED 1 (default)**

  Enables dynamic forms support. If you don't need any dynamic forms, simply set HTTP_DYNINFO_ENABLED 0. This will dramatically reduce RAM/ROM usage up to 50%.

  Please note: The HTTP_DYNINFO structure only exists if dynamic forms support is enabled.

- **HTTP_DYNINFO_SIZE_CTRNAME 16 (default)**

  This is the maximum size of the control name entry at the HTTP_DYNINFO structure which is posted to the application when the F-Server finds a form control that can be changed by the application.

  If you definitely know that all control names are shorter than 16, you can decrease this value. The F-Server will throw an error event (to the application) if control name string does not fit.

  Please note: RAM usage will be HTTP_DYNINFO_SIZE_CTRNAME +1 byte string terminator multiplied with number of HTTP_MULTI instances!

- **HTTP_DYNINFO_SIZE_PARAM1 16 (default)**
- **HTTP_DYNINFO_SIZE_PARAM2 16 (default)**
- **HTTP_DYNINFO_SIZE_PARAM3 16 (default)**
- **HTTP_DYNINFO_SIZE_PARAM4 16 (default)**

  These are the maximum sizes of the param1..4 entry (+ string terminator) at the HTTP_DYNINFO structure, which is posted to the application when the F-Server finds a form control that can be changed by the application. You can adapt these buffer sizes to your application for any parameter separately, which should always decrease your memory requirements.

Please note: RAM usage will be HTTP_DYNINFO_SIZE_PARAMx +1 byte string terminator multiplied with number of HTTP_MULTI instances!

- **HTTP_MAX_PAGENAME_LEN 16 (default)**

  Usually, web servers identify their content by names and also the F-Server does. Because the F-Server does not have a file system (it neither needs one), some code is implemented which simulates a file system's 'find-a-file' functionality.

  In order to find a file, the F-Server needs to copy the file (or resource) name from a partial HTTP request to internal memory. With HTTP_MAX_PAGENAME_LEN one can define the maximum size of that array. Names, which length doesn't fit, are truncated.

  Please note: RAM usage will be HTTP_MAX_PAGENAME_LEN +1 byte string terminator multiplied with number of HTTP_MULTI instances!

- **HTTP_QUESIZE 15 (default)**

  When the F-Server generates output, it fills a queue with fragments of HTTP data. Each queue entry occupies six bytes (four bytes holds a pointer and two bytes for length information) and is able to describe 65520 bytes of data.

  This value has to be set according to the biggest HTML resource the F-Server has to send. The default value is able to suuport 15*65.520 = 982.800 bytes.

  Unless you disable dynamic forms or have very large resources, HTTP_QUESIZE should not be changed. If dynamic forms are not needed, HTTP_QUESIZE can be decreased to four (three for HTTP headers and one for the resource) which saves 66 bytes of RAM.

  Because each entry can point to ~64 kBytes, the F-Server automatically allocates queue entries from its free pool if the resource doesn't fit into a single one. For example: If an F-Server should transmit an jpeg-image having a size of 200,000 bytes, it needs four queue entries for the entire image. According to the description above, there are a total of seven queue entries needed for the transmission of such a big picture.

- **HTTP_MAX_CTRLNAME_LEN 16 (default)**

  While processing an HTTP POST request, the F-Server extracts form control names and values into an internal buffer. HTTP_MAX_CTRLNAME_LEN is the size of a the buffer where control names are stored.

  Please note: RAM usage will be HTTP_MAX_CTRLNAME_LEN +1 byte string terminator multiplied with number of HTTP_MULTI instances!

- **HTTP_MAX_CTRLVALUE_LEN 16 (default)**

  Likewise, when processing an HTTP POST request, the F-Server extracts form control values into another internal buffer. HTTP_MAX_CTRLVALUE_LEN is the size of a the buffer where those control values are stored.

  Please note: RAM usage will be HTTP_MAX_CTRLVALUE_LEN +1 byte string terminator multiplied with number of HTTP_MULTI instances!

- **HTTP_FILEUPLOAD_ENABLED 0 (default)**

Enable the file upload feature by setting #define HTTP_FILEUPLOAD_ENABLED to 1. File upload needs 167 Bytes of additional RAM for buffers and variables on every instance. It shares RAM with form control buffers to save RAM. So you can increase HTTP_MAX_CTRLVALUE_LEN and HTTP_MAX_CTRLNAME_LEN to 162 (sum) without increasing RAM usage.

Please note: RAM usage will be 167 byte multiplied with number of HTTP_MULTI instances!

- **HTTP_SPEEDBUFFER_SIZE 256 (default)**

  Because the F-Server may produce small chunks of data when generating dynamic form controls, some embedded TCP implementations (including sevenstaxTCP), which has no send buffer, will transmit them immediately. This results in very low transmission speed because network packets are poorly utilized. To speed up the F-Server, a little send buffer can be activated through HTTP_SPEEDBUFFER_SIZE. The buffer is shared by all instances of the F-Server.

  If you don't need it (because you have a buffering transport protocol), set HTTP_SPEEDBUFFER_SIZE to 0.

  By using this this buffer, code size increases and you need HTTP_SPEEDBUFFER_SIZE more RAM bytes.

- **HTTP_AUTH_ENABLED 1 (default)**

  This is the main switch to control the F-Server's behaviour on authentication. If **HTTP_AUTH_ENABLED 1** is set, the F-Server supports HTTP authentication.

  Wether the F-Server verifies user name and password automatically or the application is to be called to verify it, is defined via **HTTP_AUTH_RAW.**

  To disable HTTP authentication, set it to 0.

- **HTTP_AUTH_RAW 0 (default)**

  This is a boolean switch which influences the F-Server's behaviour on authentication.

  If **HTTP_AUTH_RAW = 0** is set, the controlling application must supply password and user name as clear-text parameters. The F-Server verifies user name and password automatically and either sends the requested file or an access-denied page.

  If **HTTP_AUTH_RAW = 1** is set, then the F-Server calls the application's callback procedure every time it wants to check out a user name / password combination. The application must then verify the base64 decoded string and return TRUE_stx or FALSE_stx.

  Each has advantage over the other. For a complete description please see the chapter: “Authentication”.

- **HTTP_AUTH_MAX_B64_LEN 32 (default)**

  When using the 'Basic HTTP Authentication Scheme', the F-Server extracts authentication information from HTTP requests into an internal buffer. The size of this buffer is indicated by HTTP_AUTH_MAX_B64_LEN and must be big enough to hold the user name, a colon(:), the password and a terminating null character ('\0') as a base64 encoded string.

If HTTP_AUTH_MAX_B64_LEN is too small, the input is truncated and an error message is sent to the application (using the callback procedure).

To omit HTTP authentication, HTTP_AUTH_MAX_B64_LEN must be set to 0. In this case the F-Server doesn't know anything about authentication which lets him accept every HTTP request. An F-Server without authentication saves ROM and rather 16 bytes of RAM.

- **HTTP_AUTH_COUNT_USER 1 (default)**

  This defines the number of different user to handled with HTTP authentification. It needs 9 bytes per user - and of course additional memory to handle user name and passwords separately.

- **HTTP_EXTTYPES 0 (default)**

  This is a boolean switch which enables extended mime types to be delivered by the webserver. If you enable **HTTP_EXTTYPES 1,** additional - but rarely used - mime types like MPEG, BMP ... are enabled (see 'httpserv.h' for details). No more RAM will be needed, but code and ROM constants will increase by about 400 bytes.

6 Application callable Interfaces

The F-Server exports a set of public interfaces which are intended to be called by a control application. This section describes them in detail.

- **void stxHTTP_Init (PROTOCOL_NOTIFY_HANDLER handler);**

  This function must be called before the F-Server can be used. It sets up all internal states and registers a handler procedure (a pointer to a function).

  The handler procedure (AKA. callback procedure) is called by the F-Server every time it has a message for the control application. For a detailed discussion of the handler procedure please see the chapter 'Application defined Callback Procedure'.

- **void stxHTTP_Receive (CONST_stx UINT8_stx *data, UINT16_stx size, UINT16_stx connection_id);**

  When the network has got data on its HTTP channel, the controlling application must call the F-Server's stxHTTP_Receive function.

  This function accepts three parameters. The first one is a pointer to memory where the data is stored, the second one contains the number of bytes (of the received data) while the last one is a so-called 'Connection - Id'.

  The Connection-Id should be an unique value which must not change while an HTTP transaction is in progress (e.g. during an open TCP connection to a web browser). When using TCP as transport layer, it is safe to use the local port number as Connection - Id.

- **void stxHTTP_TransportLayerEvent (HTTP_TLEVENT ev, UINT16_stx connection_id);**

  This function must be called by the controlling application, when a network event occurs which the F-Server should know. It accepts two parameters. The first parameter is an HTTP_TLEVENT enumerated type and can be one of the following:

  - **HTTP_TLE_OPEN** means...

    A new connection is established and data will soon arrive. A new Connection-Id first appears here. The F-Server uses the Connection-Id to activate an instance of itself.

  - **HTTP_TLE_CLOSE** means...

    The connection is closed, regardless of a regular close or an error condition. The Connection-Id will be used to invalidate the current F-Server instance.

  - **HTTP_TLE_TXREADY** means...

    The transport layer is able to accept data for transmission. This event tells the F-Server that a previous send request has completed.

  The second parameter is the Connection-Id mentioned above.

- **void stxHTTP_Tick (void)**

  Although the F-Server is mainly driven by calls to stxHTTP_Receive(), it sometimes needs processor time to do deferred tasks such as flushing its send queue. For this purpose the controlling application must call stxHTTP_Tick() repeatedly with a minimum interval of 1 second.

  For maximum performance and if many dynamic elements are used, we recommend to handle this stxHTTP_Tick() like all other sevenstax stx_xxxTick() functions: Please call it as often as possible! If there is nothing to do, it will return immediately and will therefore not be a significant load for the user system.

- **UINT8_stx stxHTTPServ_AuthAddUser (STRING_stx szUser, STRING_stx szPwd)**

  If HTTP Autentication is enabled, this function is to be used to add a user name and password to the list of user. For details, please see 13 Authentication.

- **void stxHTTPServ_AuthAssignGroupToUser (UINT8_stx uUserIndex, UINT8._stx uGroupIndex)**

  If HTTP Autentication is enabled, this function is to be used to give a user rights of a certain group. For details, please see 13 Authentication.

7 Data Structures and Enumerated Types

This chapter describes all the data structures and enums used by the F-Server. The definitions can be found in 'http.h'.

HTTP_CONTROLTYPE

```
typedef enum
{
        HTTPC_SELECTBOX = 0,
        HTTPC_OPTION = 1,
        HTTPC_SELECTBOX_END = 2,
        HTTPC_CHECKBOX = 3,
        HTTPC_RADIO = 4,
        HTTPC_TEXTAREA = 5,
        HTTPC_RAWTEXT = 6,
        HTTPC_TEXTAREA_END = 7,
        HTTPC_TEXTFIELD = 8,
        HTTPC_SUBMIT = 9,
        HTTPC_RESET = 10,
        HTTPC_PASSWORD = 11;
        HTTPC_NOTFOUND = 101,        /* No control found */
        HTTPC_INVALID = 102          /* Only end marker */
} HTTP_CONTROLTYPE;
```

This enumeration is used to distinguish between form controls. The values belong to these types of controls:

- **HTTPC_SELECTBOX**

  A HTML select box begin tag which means something like <SELECT NAME =....>.
- **HTTPC_OPTION**

  A select box entry which means something like <OPTION ...>.
- **HTTPC_SELECTBOX_END**

  A HTML end of select box tag which is </SELECT>.
- **HTTPC_CHECKBOX**

  A HTML check box tag which means something like <INPUT TYPE = „CHECKBOX...>.
- **HTTPC_RADIO**

  A HTML radio button which means something like <INPUT TYPE = „RADIO...>.
- **HTTPC_TEXTAREA**

  A HTML text area begin tag which means something like <TEXTAREA NAME =“...>
- **HTTPC_RAWTEXT**

  A raw string of characters which may or may not include HTML.

- **HTTPC_TEXTAREA_END**

  A HTML text area end tag is </TEXTAREA>.

- **HTTPC_TEXTFIELD**

  A HTML text field which means something like <INPUT TYPE = „TEXT...>.

- **HTTPC_SUBMIT**

  A HTML submit button which means something like <INPUT TYPE = „SUBMIT...>.

- **HTTPC_RESET**

  A HTML reset button which means something like <INPUT TYPE = „RESET...>.

- **HTTPC_PASSWORD**

  A password filed which is something like <INPUT TYPE = “PASSWORD...>

- **HTTPC_NOTFOUND**

  A value only intended for internal use.

- **HTTPC_INVALID**

  This marks the end of the enumeration and is never used.

HTTP_DYNINFO

```
typedef struct _tag_HTTP_DYNINFO
{
        HTTP_CONTROLTYPE type;
        char controlname[HTTP_DYNINFO_SIZE_CTRNAME];
        char param1[HTTP_DYNINFO_SIZE_PARAM1];
        char param2[HTTP_DYNINFO_SIZE_PARAM2];
        char param3[HTTP_DYNINFO_SIZE_PARAM3];
        char param4[HTTP_DYNINFO_SIZE_PARAM4];
} HTTP_DYNINFO;
```

This structure carries information about a single form control, which is about to be created. Whenever the F-Server reads the template and finds a dynamic control entry, it builds a HTTP_DYNINFO structure and passes it to the application. The application now has the possibility to modify the form control before the F-Server generates an HTML page and transmits it to the client. The application might change the name, type and some (or all) parameters or left them all unchanged (In this case, the defaults are used).

Members of HTTP_DYNINFO have the following meanings:

- **type**

  Any value from HTTP_CONTROLTYPE enumeration. The application should read (but rarely modify e.g. to change visual appearance) this member to recognise the type of a form control.

- **controlname**

  This is the name of the form control respectively the same string as in the HTML tag 'NAME=“XXX“. The application should read this value in order to identify the form control by name. Modification of this member is also possible but not very useful.

- **param1...param4**

  These additional strings depend on the control type and its attributes. For example: A radio button can have a name, a value, a label and a flag whether it is checked or not. That is: Value, label and checked-flag are stored in param1, param2 and param3 (param4 is not used for radio buttons).

HTTP_POSTINFO

```
typedef struct _tag_HTTP_POSTINFO
{
    STRING_stx name;        /* Name of control */
    STRING_stx value;       /* Value as string */
} HTTP_POSTINFO;
```

This simple structure only contains two pointers to zero-terminated strings of variable length. When a HTTP POST arrives, the F-Server analyses and decodes the content and (if it finds messages from form controls) sets up an HTTP_POSTINFO structure which is delivered to the application. The meanings of both members are:

- **name**

  This is the name of the form control which was transmitted by the HTTP POST command. The application may do a string compare on this member to identify a certain form control.

- **value**

  This is the new value of the form control named 'name'. If it came from a text field which contains a numeric value, the application may convert this string into a number and process it some way.

Please note: The application must not change the contents of both members (they are meant for being read-only).

HTTP_MPDATA

```
typedef struct _tag_HTTP_MPDATA
{
    UINT16_stx size;            /* size of data packet/buffer */
    UINT8_stx FPTR_stx data; /* pointer to data/buffer */
} HTTP_MPDATA;
```

On file upload a NC_HTTP_POSTDATA_FILE or NC_HTTP_POSTDATAEND_FILE event will be generated. To deliver the received file data to the application a HTTP_MPDATA structure will be used. The meanings of both members are:

- **data**

  Pointer to received data.

- **size**

  number of received data bytes.

Please note: The application must not change the contents of both members (they are meant for being read-only).

HTTP_TLEVENT

```
typedef enum
{
        HTTP_TLE_TXREADY,          /* TCP is ready to send or has tx'ed a packet */
        HTTP_TLE_OPEN,             /* TCP connected */
        HTTP_TLE_CLOSE,            /* TCP disconnected */
        HTTP_TL_INVALID            /* End marker, do not use */
} HTTP_TLEVENT;
```

HTTP_TLEVENT is a enumeration for use as the first parameter to stxHTTP_TransportLayerEvent() (Please see the chapter: Application callable interfaces). The values have the following meanings:

- **HTTP_TLE_OPEN**

  A new connection is established and data will soon arrive.

- **HTTP_TLE_CLOSE**

  The connection is closed, regardless of a regular close or an error condition.

- **HTTP_TLE_TXREADY**

  The transport layer is able to accept data for transmission.

HTTP_ERROR

```
typedef enum
{
        HTTPERR_AUTHSTR_OVERFLOW,
        HTTPERR_PAGENAME_OVERFLOW,
        HTTPERR_CTRLVALUE_OVERFLOW,
        HTTPERR_CTRLNAME_OVERFLOW,
        HTTPERR_DYNINFO_OVERFLOW,
        HTTPERR_QUEUE_OVERFLOW,
        HTTPERR_CTRLBOUNDARY_OVERFLOW,
        HTTPERR_INVALID                          /* End marker, do not use */
} HTTP_ERROR;
```

This is an enumeration type, which is used, when the F-Server has detected an error condition from which it can't recover. The F-Server then calls the application defined handler procedure with one of the members telling the application what's going wrong.

These error codes have following meanings:

- **HTTPERR_AUTHSTR_OVERFLOW**

  This error code is thrown when the base64 encoded password / user name combination doesn't fit into the supplied buffer. The F-Server extracts this information from HTTP requests by a call to stxHTTP_Receive(). The size of the buffer must be #defined before

compilation by HTTP_AUTH_MAX_B64_LEN. With a simple formula below one can calculate the necessary space:

*HTTP_AUTH_MAX_B64_LEN = 4 *(2+(1+strlen(user)+strlen(pwd))/3))*

- **HTTPERR_PAGENAME_OVERFLOW**

  This error code is posted to the callback procedure when the page (or resource) name doesn't fit into the buffer which size must be set by HTTP_MAX_PAGENAME_LEN. The page name is right part of an URL e.g. http://192.168.130.130/page name

- **HTTPERR_CTRLVALUE_OVERFLOW**

  This error code is posted to the callback procedure when a form control's **value** is too big to fit into the supplied buffer. The F-Server extracts form control values from HTTP POST requests. The maximal length which the F-Server accepts must be #defined by HTTP_MAX_CTRLVALUE_LEN.

- **HTTPERR_CTRLNAME_OVERFLOW**

  This error code is posted to the callback procedure when a form control's **name** is too big to fit into the supplied buffer. The F-Server extracts form control names from HTTP POST requests. The maximal length which the F-Server accepts must be #defined by HTTP_MAX_CTRLNAME_LEN.

- **HTTPERR_QUEUE_OVERFLOW**

  This error code is posted to the callback procedure when there are no more free entries in the transmit queue. This mostly occurs, when large files are about to be sent which need many queue entries. If an HTTPERR_QUEUE_OVERFLOW appears, you should increase the number of queue entries which is #defined by HTTP_QUESIZE.

- **HTTPERR_DYNINFO_OVERFLOW**

  This error code is thrown when the name or one of the parameter strings doesn't fit into the HTTP_DYNINFO structure. To avoid this error, shorten the name or parameter value of the specific form control. This means the template needs a little redesign. If you need such long names or parameter values, you must increase HTTP_DYNINFO_BUFFERSIZE.

- **HTTPERR_CTRLBOUNDARY_OVERFLOW**

  This error code is thrown when file upload is enabled and internal buffers can not handle a incoming file. If this happens the sender of file could be using a non standard format. Check the sender in this case. File size does not matter the internal buffers.

- **HTTPERR_INVALID**

  This is used as end marker only. It will never appear as error code.

HTTP_AUTH

```
typedef struct _tag_HTTP_AUTH
{
        STRING_stx user;          /* User name */
        STRING_stx pwd;           /* Password */
} HTTP_AUTH;
```

This structure is for internal use only (but documented here for completeness). It only exists when authentication is enabled. HTTP_AUTH carries the following members:

- **user**

  This is the user name (a character pointer). It points to the user name and a terminating null character.

- **pwd**

  This is the password (also a character pointer).

  Both pointers are set by a call to stxHTTP_SetupAuthentication().

HTMLP_TYPE

```
typedef enum
{
        HTYPE_HTML,
        HTYPE_DYNHTML,
        HTYPE_GIF,
        HTYPE_JPEG,
        HTYPE_JAVA_APPLET,
        HTYPE_RAW,
        HTYPE_INVALID          /* End marker, do not use */
} HTMLP_TYPE;
```

HTMLP_TYPE is an enumeration which is used to distinguish between different resource types. The browser (and the F-Server also) wants to know what kind of data comes along.

A HTMLP_TYPE enumeration is part of the file descriptor (see HTMLP_FILE below) and has the following members:

- **HTYPE_HTML**

  This means that the file is an ordinary HTML page with doesn't need extra attention. The F-Server constructs a HTTP header with 'Content-Type: text/html' and sends off the file to the client (browser).

- **HTYPE_DYNHTML**

  A file which has this attribute is a dynamic HTML form template. If the browser requests such a file (by name), the F-Server parses the file searching for special dynamic HTML tags and exchanges those tags with dynamic data which it queries from the application.

From the browser's view, a file of type HTYPE_DYNHTML looks like any other HTML form. This also means that the F-Server inserts 'Content-Type: text/html' into the HTTP header.

- **HTYPE_GIF**

  This marks the file as a 'gif-image' The F-Server writes a 'Content-Type: image/gif' - entry into the HTTP header and then sends the file without observance.

- **HTYPE_JPEG**

  The same handling as above (gif image) but this marks a jpeg-image.

- **HTYPE_RAW**

  This type is intended for data transfer from the F-Server to a computer without being displayed in a web browser's window. Coming from the F-Server, a web browser will see such resources as 'application/x-raw-stuff'. Browsers receiving a file of type 'application/x-raw-stuff' should open a file-save dialogue and let the user save the content to disk.

- **HTYPE_JAVA_APPLET**

  If a file has this attribute, the F-Server constructs a 'Content-Type' which is 'application/x-Java-applet;version=1.1.' A web browser needs this to activate its JAVA virtual machine (if it has one). The subsequent data part must be a JAVA program.

- **HTYPE_INVALID**

  This is only an end marker of the enumeration and will never appear in the source code.

HTMLP_FILE

```
typedef struct _tag_HTMLP_FILE
{
        UINT8_stx PTR rname;
        HTMLP_TYPE type;
        UINT8_stx PTR data;
        UINT32_stx datasize;
} HTMLP_FILE;
```

This structure is used to describe a single resource (e.g. HTML page or picture).

When the client (web browser) requests a file, the F-Server searches an internal array of HTMLP_FILE structures (by name compare) until it finds a matching one. The structure holds all information necessary for the F-Server to identify, change and send a resource to the client.

A HTMLP_FILE structure has the following members:

- **name**

  This member points to name of the file (e.g. web page or other resources). The name must be a regular c-string including a terminating NULL-character.

  If the site should appear in a structured tree-like manner, the name must have all components of a path name. For example: If a browser sends 'GET /peter/ebooks/manual.sxw', the name of the resource must be the full path '/peter/ebooks/manual.sxw'<0>' (the <0> means a single zero after ...sxw). A start page should have the name '/' because this is what web browsers send in an GET request, when no explicit resource is requested.

- **type**

Every resource needs a type qualifier because the F-Server must know how to construct a proper HTTP header and also if the resource needs modification. The type member is a HTMLP_TYPE enumeration, which is described above (see HTMLP_TYPE).

- **data**

  This member contains the memory address of the first byte of the resource.

- **datasize**

  This is the size (in bytes) of the resource.

8 How the F-Server accesses files and other resources

While the F-Server is mainly targeted on embedded systems, it also can be used on bigger platforms like desktop PCs and server machines. On these machines, it doesn't make much sense to keep all pages in memory because they have fast hard disk drives.

To access resources for transmission, the F-Server only relies upon three interface calls. The default implementation (html_page.c) is a file system simulation which maps huge arrays (containing the pages) to HTMLP_FILE structures.

Because of its simplicity, the F-Server's virtual file system is very easy to exchange. Here is a description of these interfaces.

Please Note: All function names regarding to file system access start with 'stxHTMLP' where the 'stx' is the sevenstax prefix and 'HTMLP' stands for 'HTML-Page'.

- **void stxHTMLP_Init (void);**

  This function is called once from the F-Server's start-up function stxHTTP_Init(). The default implementation initializes all HTMLP_FILE structures (each for every single page) and the HTMLP_FILE structure for the 'not-found' page.

- **HTMLP_FILE PTR stxHTMLP_Find (STRING_stx name);**

  The F-Server calls this function in order to get a file descriptor (HTMLP_FILE pointer) by name. The default implementation iterates through the array until it finds the matching name. If there's no match, the user is asked to deliver a dynamically generated file. This is done with NC_HTTP_USERFILE. If the user has no dynamic file with this name the 'not-found' page will be displayed.

- **void stxHTMLP_Release (HTMLP_FILE PTR filedesc);**

  This function is called, when the F-Server has transmitted the file and does not need it any more. The default implementation does nothing. Because all files are static data, there's no need to free anything.

The listing below shows a complete and fully functional virtual file system for MS-Windows.

```
/* Get windows headers */
#include <windows.h>

/* General includes */
#include "features.h"
#include "stxtypes.h"
#include "ipv4.h"
#include "notifycodes.h"
#include "debug.h"
#include "protdefs.h"
#include "tcp.h"
#include "httpserv.h"
#include "runtimelib.h"

/* Holds 'page not found' resource */
HTMLP_FILE HTMLP_notfound;
static CONST_stx UINT8_stx HTMLP_nf_data[] = "<HTML><HEAD><TITLE>7stax
webserver</TITLE></HEAD><DIV ALIGN=CENTER><FONT SIZE=+4><BR><BR><BR>Sorry, the requested
file does not exist on this server.</FONT></DIV></HTML>";
```

```
/* Free file descriptors */
void stxHTMLP_Release (HTMLP_FILE PTR filedesc)
{
	/* Only free them if they are valid ... */
	if (filedesc)
	{
		/* ... and not the 'empty page' descriptor */
		if (filedesc->data != HTMLP_nf_data)
		{
			if (filedesc->data)
				free (filedesc->data);
			free (filedesc);
		}
	}
}

/* Given the name, find file descriptor */
HTMLP_FILE PTR stxHTMLP_Find (STRING_stx name)
{
	/* Some strings */
	char fullname[MAX_PATH];
	char tmp[MAX_PATH];
	char dir[_MAX_DIR];
	char ext[_MAX_EXT];

	HTMLP_FILE PTR filedesc = 0;
	HANDLE hfile = INVALID_HANDLE_VALUE;
	DWORD bytes_read;

	/* Make full path name */
	GetModuleFileName (NULL, tmp, MAX_PATH);
	_splitpath (tmp, fullname, dir, 0, 0);
	strcat (fullname, dir);
	strcat (fullname, "wwwroot\\");
	if (name[0] == 0)
		strcat (fullname, "index.html");
	else
		strcat (fullname, name);

	/* Now open the file */
	hfile = CreateFile (fullname,
			GENERIC_READ,
			0, NULL,
			OPEN_EXISTING,
			0,
			 NULL);
	/* Not found */
	if (hfile == INVALID_HANDLE_VALUE)
		goto out_on_error;

	/* Make a new file descriptor */
	filedesc = (HTMLP_FILE PTR)malloc (sizeof(HTMLP_FILE));
	/* No more heap space */
	if (filedesc == 0)
		goto out_on_error;

	/* Get the size and malloc memory for the file data */
	filedesc->datasize = GetFileSize(hfile, 0);
	filedesc->data = (UINT8_stx PTR)malloc (filedesc->datasize);
	/* No more heap space */
	if (filedesc->data == 0)
		goto out_on_error;

	/* Read in the file  and close it */
	if (FALSE_stx == ReadFile (hfile, filedesc->data, filedesc->datasize, &bytes_read, 0))
		goto out_on_error;
	CloseHandle (hfile);

	/* Get the type from resource's extension string */
```

```
	_splitpath (fullname, 0, 0, 0, ext);
	if (strcmpi(ext, ".html") == 0)
		filedesc->type = HTYPE_HTML;
	else if (strcmpi(ext, ".fhtml") == 0)
		filedesc->type = HTYPE_DYNHTML;
	else if (strcmpi(ext, ".gif") == 0)
		filedesc->type = HTYPE_GIF;
	else if (strcmpi(ext, ".jpg") == 0)
		filedesc->type = HTYPE_JPEG;
	else if (strcmpi(ext, ".jpeg") == 0)
		filedesc->type = HTYPE_JPEG;
	else if (strcmpi(ext, ".class") == 0)
		filedesc->type = HTYPE_JAVA_APPLET;
	else
		filedesc->type = HTYPE_RAW;

	/* Give the new descriptor to the app */
	return filedesc;

	/* error exit */
	out_on_error:
	stxHTMLP_Release (filedesc);
	if (hfile != INVALID_HANDLE_VALUE)
		CloseHandle (hfile);
	return &HTMLP_notfound;
}

/* Build all the file descriptors */
void stxHTMLP_Init (void)
{
	/* Initialize the 'notfound' reply */
	HTMLP_notfound.rname = 0;
	HTMLP_notfound.data = (UINT8_stx PTR)HTMLP_nf_data;
	HTMLP_notfound.datasize = (UINT32)sizeof (HTMLP_nf_data)-1;
	HTMLP_notfound.type = HTYPE_HTML;
}
```

9 Application defined Callback Procedure

When the F-Server has something to report e.g. it has data to transmit or decoded an HTTP POST message, the application will receive such events by a dedicated callback routine (aka handler procedure).

The handler procedure must be registered with the F-Server by a call to stxHTTP_Init(). Currently, there are seven different events which the F-Server may throw.

A handler procedure should look like this:

```
UINT32_stx handlerproc (NOTIFY_CODE a, UINT32_stx b, UINT16_stx c)
{
	switch (a)
	{
		/*
			Notify web server when TCP is connected
		*/
		case NC_TCP_CONNECTED:
		stxHTTP_TransportLayerEvent (HTTP_TLE_OPEN, c);
		break;

		/*
			Received data on port 80 is transferred to the web server
		*/
		case NC_TCP_RECV:
		stxHTTP_Receive (((TCP_INFORMATION*)b)->data, ((TCP_INFORMATION*)b)->datasize, c);
		break;

		/*
			Tell web server that the transmitter is ready
		*/
		case NC_TCP_TXREADY:
		stxHTTP_TransportLayerEvent (HTTP_TLE_TXREADY, c);
		break;

		/*
			Notify web server when TCP closed a connection
			and go to listen state again.
		*/
		case NC_TCP_FIN:
		stxHTTP_TransportLayerEvent (HTTP_TLE_CLOSE, c);
		stxTCP_Listen (80);
		break;

		/*
			Let the app authenticate.
			The base64 decoded string is 'peter:lieb'
		*/
		case NC_HTTP_AUTH:
		if (strcmp (b, "cGV0ZXI6bGllYg==") == 0)
			return TRUE_stx;
		return FALSE_stx;

		/*
			The web server wants to send some data.
			TCP must be called to perform this.
		*/
		case NC_HTTP_TX:
		{
			HTTP_TXDATA *tx = (HTTP_TXDATA*)b;
			return (UINT32)stxTCP_WriteBulk (tx->data, tx->size, c);
		}
		break; /* Never gets here */

		/*
```

```
                        A HTTP transaction has completed.
                        The TCP connection must now be closed.
                */
                case NC_HTTP_COMPLETE:
                stxTCP_Close (c);
                break;

                /*
                        The user has pushed the submit button
                */
                case NC_HTTP_POSTINFO:
                ProcessPostInfo ((HTTP_POSTINFO PTR)b);
                break;

                /*
                        Web server wants to build dynamic controls.
                        An application can now change them.
                */
                case NC_HTTP_DYNINFO:
                ProcessDynInfo ((HTTP_DYNINFO PTR)b);
                break;

                /*
                        Web server is in serious trouble.
                */
                case NC_HTTP_ERROR:
                _asm hlt;
                break;
        }
        return 0L;
}
```

The F-Server calls the handler procedure with three parameters, a NOTIFY_CODE (which is a 16 bits value where the upper 8 bits are reserved, notice the '&0xff') and two values, one is 32 (long parameter) and one is 16 (short parameter) bits of size. The first parameter (NOTIFY_CODE) stands for a certain event. The other two contain additional information on this event.

Some events need a return value from the handler procedure to give the caller (the F-Server) a feedback e.g. if an action succeed or not. For events which does not need a return value, the handler procedure should return zero.

A little care must be taken, while staying in the handler procedure:

- Don't waste much time here, except on NC_HTTP_BUSY.
- Avoid calls into the F-Server except those which are explicitly allowed. Please see description of functions for details.
- Keep in mind that all arguments of the handler procedure (NOTIFY_CODE, parameters) are of temporary nature and lose validity on exit.

Detailed description about the different events:

- **NC_HTTP_COMPLETE**

  This event will be produced, when the F-Server has completed an HTTP transaction. An HTTP transaction is the summary of all actions which are necessary to satisfy a single HTTP request.

  When the application receives this event, it can be sure that the F-Server is ready and may close the TCP connection. The long parameter (l_param) is not used. The short parameter (s_param) contains the current Connection ID. A return value is not used.

- **NC_HTTP_POSTINFO**

  The F-Server issues one or more of such events, when the client side (web browser) has sent data with an HTTP POST command. For every single form control, the F-Server builds a HTTP_POSTINFO structure, which carries information about the form control (name and value) and delivers it to the application by using the long parameter (l_param) as a pointer to the structure.

  This pointer and the structure members must not be changed and are read-only. Also pointer, structure members and their content are only valid inside the handler procedure. If you need them later, you should copy the content to another location.

  The short parameter (s_param) contains the current Connection ID, a return value is not used.

- **NC_HTTP_DYNINFO**

  While the F-Server is sending a form template to the browser, it searches for its own special dynamic HTML tags which are embedded into the template. If it finds such a tag, the F-Server builds a HTTP_DYNINFO structure and passes it to the application.

  Members of that structure contain default values, which are gained from the template. The application now can change these values or leave them alone.

  The long parameter (l_param) contains a pointer to a HTTP_DYNINFO structure while the short parameter (s_param) carries the Connection ID.

  Generally the F-Server expects a Zero return value, if the application did successfully handle the DYNINFO call. If your application is not yet ready to insert the requested dynamic information (e.g. the requested value is not yet available, but will be soon), your application call back function should return a Non-Zero value. This will force the F-Server to repeatedly request this DYNINFO, until the application returns Zero for it. Please assure a timeout handling in your application callback function, if you use this feature.

- **NC_HTTP_ERROR**

  When the F-Server detected an error condition, it posts an NC_HTTP_ERROR event to the application. In most cases, NC_HTTP_ERRORs result from a configuration failure which can be corrected by compile-time options or form template redesign.

  If such an error occurs, the entire HTTP transaction has failed but it is possible that the application receive other events after that error (when it feeds the F-Server further on). To avoid this, the application should stop the F-Server e.g. immediately close the network connection when it gets an NC_HTTP_ERROR.

The long parameter (l_param) informs the application about the kind of error. For further information about possible errors (HTTP error enumeration) please see the chapter 'Data Structures and Enumerated Types'.

The short parameter (s_param) contains the connection ID.

A return value is not used and should be zero.

- **NC_HTTP_TX**

  Because the application is responsible for sending (and receiving) over the network, the F-Server posts this event, when it has data which must be sent.

  The application receives a pointer (as the long parameter 'l_param')to an HTTP_TXBUFFER structure, where it can read the 'data' and 'size' members describing location and number of bytes which it must deliver to the transport protocol for transmission. Please see the chapter 'Data Structures and Enumerated Types' for a detailed description of the HTTP_TXBUFFER structure.

  The short parameter (s_param) contains the connection ID.

  The F-Server needs to know if transmission was successful or not. Therefore, the applications must return a non-zero value as result of a successful send. Otherwise it should return zero to notify the F-Server of a send failure.

- **NC_HTTP_AUTH**

  This event will be posted only if raw authentication is enabled (HTTP_AUTH_MAX_B64_LEN>0 **and** HTTP_AUTH_RAW =1). Every time the F-Server gets a base64 encoded password / user name combination, it calls the callback procedure in order to let the application validate them.

  The long parameter (l_param) holds a pointer to a base64 encoded, null-terminated string with the format 'USERNAME:PASSWORD'. The application should check this and must return the result which means: TRUE_stx (access granted) or FALSE_stx (access denied).

- **NC_HTTP_POSTINFO_FILE**

  If file upload is enabled and a file arrives the F-Server posts an NC_HTTP_POSTINFO_FILE event. A HTTP_POSTINFO structure, which carries information about the file (variable and file name), delivers information to the application by using the long parameter (l_param) as a pointer to the structure.

  This pointer and the structure members must not be changed and are read-only. Also pointer, structure members and their content are only valid inside the handler procedure. If you need them later, you should copy the content to another location.

  Variable name is stored at HTTP_POSTINFO.name and file name is stored at HTTP_POSTINFO.value.

  The short parameter (s_param) contains the current Connection ID.

  A return value is not used.

- **NC_HTTP_POSTDATA_FILE**

  After signaling the NC_HTTP_POSTINFO_FILE event, the according files content is to be given to the application. Therefore the F-Server posts an NC_HTTP_POSTDATA_FILE

event to the callback procedure for every file part. The long parameter (l_param) contains a pointer to a HTTP_MPDATA structure, where the received file data can be accessed.

A return value is not used.

- **NC_HTTP_POSTDATAEND_FILE**

  If the last part of uploading file was received, a NC_HTTP_POSTDATAEND_FILE event is generated. The long parameter (l_param) contains a pointer to a HTTP_MPDATA structure, where the last received file data packet can be accessed.

  A return value is not used.

- **NC_HTTP_USERFILE,**
  NC_HTTP_GETUSERFILELEN

  If the F-Server can't find a requested filename in the resources the user will be asked via NC_HTTP_USERFILE if he has a dynamic file with this filename.

  param1 points to a HTMLP_FILE structure, param2 has the connection id.
  In the first call the 'rname' member of the HTMLP_FILE structure points to the requested filename. If the user has dynamic data he has to modify the member 'data' so that it points to the data segment. The member 'datasize' must indicate the size of the data segment. If so the user will be asked for the length of the complete file via NC_HTTP_GETUSERFILELEN where param1 points to a UINT32_stx variable where the user can put his answer, param2 has the connection id. When the value is greater than 0 a 'Content-Length' entry will be inserted.

  The calls of NC_HTTP_USERFILE are repeated until the user returns NULL_stx in the 'data' member. If this is the case in the first call, the 'not found' page is displayed.

- **NC_HTTP_PAGENAME**

  Straight before the F-Server sends a page, it posts an NC_HTTP_PAGENAME event to the callback procedure. The long parameter (l_param) contains a pointer to F-Server's internal memory where it stores the name of the resource to send (as a zero-terminated string). By modifying these bytes, one can force the F-Server to send out another file.

10 Creating Resources (HTML Files, Bitmaps, etc.)

Resources are files (e.g. bitmaps or HTML pages) which the F-Server sends to the browser as a result of some HTTP requests.

As mentioned above, to store those resources, the F-Server does not have (and does not need) a file system. It uses an array of HTMLP_FILE structures which has all information on the resources.

The F-Server manages his resources through 'html_page.c'. The resources are located in 'website_rsc.c' which can easily be generated by the delivered tool 'RessourceGen.exe'.

Currently, the F-Server knows about these resource types:

- HTML pages
- Dynamic HTML forms
- GIF images
- JPEG images
- JAVA applets
- RAW data (like binarys)

Resources usually begin their life on desktop computer systems running dedicated programs like HTML editors and image tools. To put a resource into the F-Server's memory, some conversion must be done. This also applies to dynamic HTML forms described in the next chapter. Here are the necessary steps which must be performed:

1. Create the resources (e.g. HTML pages) and put them to a directory. Subdirectorys are supported. Be sure the start file (the web page mostly called 'index.html') of the web pages is not located in subdirectory.
2. Run the standard accessory utility 'RessourceGen.exe' and select the start file from step 1. After you clicked the 'OK' button, two new files will be created in the same directory where the tool 'RessourceGen.exe' is located. The files are named 'website_rsc.c' and 'website_rsc.h'.
3. Copy the newly generated files 'website_rsc.c' and 'website_rsc.h' to your project folder and rebuild your project. Now your resource is ready to be served.

11 Creating dynamic HTML Forms

As written above, the F-Server's biggest strength is its ability to create dynamic HTML pages at runtime. For this purpose, the F-Server understands an unique 'language' which must be embedded into HTML forms. This chapter focuses on this and also does a little description on HTML forms in general.

HTML forms are HTML documents (or parts of a document), where users can enter information and send it back to the source. If the browser found a certain HTML tag '<FORM...>' it is prepared to render so-called 'form controls' which can be buttons, menus, text fields, check boxes and so on. The user may complete the form e.g. by entering some text and then send the form back to the web server. There is a special purpose button (the 'submit' button) which instructs the browser to send off the form inclusively user input. This is performed by an HTTP POST request.

As already told, a form begins with an HTML 'FORM' tag. There are some additional options one can insert into a FORM tag (e.g. METHOD, ACTION and TARGET). To ensure best behaviour of the F-Server, the only option should be METHOD=“POST“. If this is true, the browser sends back the form to the F-Server, which will process it and replies with the form again (to reflect changed values). The following shows an empty HTML page which can be a starting point, when you write templates for the F-Server:

```
<!DOCTYPE HTML PUBLIC „-//W3C//DTD HTML 3.2 //EN“>
<HTML>
<HEAD>
<TITLE>
This line is displayed in the browser's title bar
</TITLE>
</HEAD>
<BODY>
<FORM METHOD=“POST“>
<!-- This is a comment -->
<!-- Some form controls and also... -->
<!-- ...dynamic tags for the F-Server... ->
<!-- ...should be here. -->
</FORM>
</BODY>
</HTML>
```

Think of this empty template page only as an advice. The F-Server does not process any HTML tags except his own dynamic ones. Please see the next table for common form controls (and options) the F-Server knows of:

(1) Type	*(2) HTML code snippet*	*(3) Supported attributes*	*(4) Dynamic ID Number*	*(5) Dynamic ID Symbol*
Selection box start tag	<SELECT...	NAME, MULTIPLE	0	HTTPC_SELECTBOX
Selection box entry	<OPTION...	SELECTED, VALUE	1	HTTPC_OPTION

(1) Type	(2) HTML code snippet	(3) Supported attributes	(4) Dynamic ID Number	(5) Dynamic ID Symbol
Selection box end tag	</SELECT>	-	2	HTTPC_SELECTBOX_END
Checkbox	<INPUT TYPE = „CHECKBOX...	NAME, VALUE, CHECKED	3	HTTPC_CHECKBOX
Radio button	<INPUT TYPE = „RADIO...	NAME, VALUE, CHECKED	4	HTTPC_RADIO
Textarea start tag	<TEXTAREA...	NAME, ROWS, COLS	5	HTTPC_TEXTAREA
Rawtext	Anything. Don't care if HTML or not.	-	6	HTTPC_RAWTEXT
Textarea end tag	</TEXTAREA>	-	7	HTTPC_TEXTAREA_END
Text field	<INPUT TYPE = „TEXT...	NAME, VALUE, SIZE, MAXLENGTH	8	HTTPC_TEXTFIELD
Submit button	<INPUT TYPE = „SUBMIT...	NAME, VALUE	9	HTTPC_SUBMIT
Reset button	<INPUT TYPE = „RESET...	NAME, VALUE	10	HTTPC_RESET

Table 11.1

The table above contains all form controls which the F-Server currently knows. In the second column (column #2) some HTML codes are shown (as code snippets) which are generated by the F-Server. The next column (#3) shows which optional attributes are supported by this current F-Server version. Column #4 contains the ID values. These values are used by the F-Server, to identify a certain control. In the last column (#5) you can see symbolic names corresponding to the ID values from column #4.

Dynamic HTML form controls, which the F-Server should process, must be embedded into an HTML page using an own specific syntax. In fact, the dynamic tags are a special kind of HTML comments, which contain all necessary information to build real form controls at runtime. A HTML comment looks like this:

```
<!-- This is a very nice comment. Isn't it? -->
```

Those comments are normally meant for humans and are not visible to a web browser's rendering engine. In contrast, the F-Server's dynamic form tags are meant for the F-Server itself and look like this:

```
<!--|09|Send|Plonk|-->
```

Everywhere in the HTML template, where this kind of comment appears, the F-Server inserts a submit button named 'Send' which is labelled 'Plonk'. Here is the HTML code generated by the F-Server.

```
<INPUT TYPE="Submit" NAME="Send" VALUE="Plonk">
```

As mentioned before, the F-Server uses its own unique syntax for generating dynamic form controls. Let's dissect the example dynamic tag from above:

(1) *Begin comment*	*(2)* *Separator*	*(3)* *ID*	*(4)* *Separator*	*(5)* *Name*	*(6)* *Separator*	*(7)* *Value*	*(8)* *Separator*	*(9)* *End comment*
<!--	|	09	|	Send	|	Plonk	|	-->

Table 11.2

All dynamic form controls must start with the sequence '<!--'. This is the beginning of an HTML comment. The next character (column #2) must be the vertical line character '|', ASCII value 124 (Hex 7c). This character is also used to separate single fields of the dynamic tag. The F-Server searches for '<!--|' to recognise a dynamic tag.

The following field (column #3) is the type (or dynamic ID) of the control as a decimal value which always must have two digits. Thus, legal values for this field are in the range '00...99' but the current version only accepts '00' to '10'. The ID is a direct representation of a control type. Please see table 11.1 for all supported dynamic control types.

After that, a separator appears (column #4) which is followed by the name field (column #5). The name field contains the name of the dynamic control which is, in most cases, transformed into 'NAME=“name“. Every dynamic control should have an unique name in order to be identified by the application. Sometimes, the name is only intended for identifying and will not generate HTML code. For example: The name of a 'Rawtext' dynamic control (ID value 06) does not appear in the generated code.

Beginning with column #7, up to four optional parameters (or options) may follow, all separated by '|"s. These parameters depend on a certain control. In this example, the submit button only needs a single parameter which is its lettering ('Plonk').

The last few letters (column #9, '-->') are the comment closing characters. They must be at the end of every dynamic control as well as at the and of every HTML comment.

Please note: Strictly stick to the syntax. If you missed out something, especially separators or the closing characters '-->', The F-Server might misbehave and unpredictable results may occur. So be careful when writing dynamic control tags.

Do you think that's confusing? Please see the next chapter for sample dynamic form controls and further explanations.

12 Dynamic Controls Examples

This chapter shows some dynamic form controls and what the F-Server produces, when it finds them:

Selection Box 1

When the template contains...
<!--|00|List1|--> <!--|01|List1|Red|Red|--> <!--|01|List1|Green|Green|--> <!--|01|List1|Blue|Blue|--> <!--|02|List1|-->
The F-Server generates...
<select NAME="List1"> <option VALUE="Red">Red <option VALUE="Green">Green <option VALUE="Blue">Blue </select>

This will display a simple selection box which has three options; Red, Green and Blue.

Selection Box 2

When the template contains...
<!--|00|List1|1|--> <!--|01|List1|Red|Red|--> <!--|01|List1|Green|Green|--> <!--|01|List1|Blue|Blue|--> <!--|02|List1|-->
The F-Server generates...
<select NAME="List1" MULTIPLE> <option VALUE="Red">Red <option VALUE="Green">Green <option VALUE="Blue">Blue </select>

This will also display a selection box which has three options. The difference between this one and the selection box before (selection box 1) is the MULTIPLE attribute. It is evoked by the '1' in the first line of the template. It doesn't matter which letter is used. Any other letter (or word) will also switch on the MULTIPLE attribute. MULTIPLE attributes enables a user to select more than one option.

Check box 1

When the template contains...
<!--|03|MyCheckBox|-->
The F-Server generates...
<input TYPE="Checkbox" NAME="MyCheckBox" VALUE="">

The browser shows only a little square (check box).

Check box 2

When the template contains...
<!--|03|MyCheckBox|Mama|Hello|1|-->
The F-Server generates...
<input TYPE="Checkbox" NAME="MyCheckBox" VALUE="Mama" CHECKED>Hello

This is displayed as a check box which is checked and which has a label at its right (Hello). Please notice the last option of the dynamic control tag. The '1' is responsible for generating a checked control but it does not matter which letter is used. To generate an unchecked control, the last option must be omitted (i.e. <!--|03|MyCheckBox|Mama|Hello|-->).

Radio Buttons

When the template contains...
<!--|04|MyRadio|Hot|Fire|--> <!--|04|MyRadio|Cold|Ice|-->
The F-Server generates...
<input TYPE="Radio" NAME="MyRadio" VALUE="Hot">Fire

```
<input TYPE="Radio" NAME="MyRadio" VALUE="Cold">Ice
```

The browser shows two radio buttons which belong to the same group (because they have the same name). Radio buttons can be checked mutually exclusive which means that only one can be checked at a time.

Text Areas

When the template contains...

```
<!--|05|MyTxtArea|40|10|-->
<!--|06|MyText|A Textbox|-->
<!--|07|MyTextAreaEnd-->
```

The F-Server generates...

```
<textarea NAME="MyTxtArea" ROWS="10" COLS="40">
A Textbox
</textarea>
```

This sequence of dynamic form tags generates a rectangular field in which the user can enter text. The text area already has some text which is generated by the second line '<!--|06|...'. Please note the different names for all three dynamic tags. Because the text areas name is the one from the first dynamic tag '<!--|05...', the other names are redundant but must be there to satisfy the F-Server's parsing rules.

Text Fields

When the template contains...

```
<!--|08|MyName|Merry X-mas and a happy new year|-->
```

The F-Server generates...

```
<input TYPE="Text" NAME="MyName" VALUE="Merry X-mas and a happy new year">
```

A text field (in contrast to text areas) can hold only a single line. In this example, the text field is already occupied with some words, which might be a hint for the user.

Submit Buttons

When the template contains...

```
<!--|09|MySubmit|Let's GO!|-->
```

The F-Server generates...

```
<input TYPE="Submit" NAME="MySubmit" VALUE="Let's GO!">
```

This dynamic form control generates a submit button. The button is labelled 'Let's GO!“.

Reset Buttons

When the template contains...

```
<!--|10|MyReset|-->
```

The F-Server generates...

```
<input TYPE="Reset" NAME="MyReset" VALUE="">
```

This dynamic form control generates a reset button.

Raw Text

When the template contains...

<!--|06|RawText|<H1>Hello World</H1>|-->
The F-Server generates...
<H1>Hello World</H1>

The F-Server is able to generate any kind of text or HTML code, not even form controls. This example shows how to insert some HTML dynamically. In the browser, the text 'Hello World' will be displayed using a big fat font.

For advanced requirements with dynamic HTML pages, like completely dynamic pages with a various number dynamic elements, RawText might be used too.

13 Authentication

The F-Server supports the so-called 'Basic HTTP Authentication Scheme'. This is a simple mechanism at what the web browser sends user name and password as base64 encoded string.

Using HTTP authentication is useful if you don't want to see any plain-text user names and/or passwords. Although, base64 is not a good encryption method, hiding plain-text may improve security a little bit.

Enable HTTP Authentication

To basically enable the F-Servers HTTP Authentication set **HTTP_AUTH_ENABLED 1** in 'httpserv.h' / 'features.h'.

If you want the F-Server to handle the HTTP Authentication automatically, set **HTTP_AUTH_RAW 0**. If you set this to 1, the F-Server will generate Callback-Function calls to request the users application to do the HTTP Authentication manually (see next paragraph).

Password Verification

The F-Server supports two different kinds of password verification depending on HTTP_AUTH_RAW (which is defined in 'httpserv.h' / 'features.h'):

- If HTTP_AUTH_RAW is set to 1, the user application gets the 'raw' base64 encyrpted data and is responsible to do the base64 password decoding and verification.
- If HTTP_AUTH_RAW is set to 0 (default), the F-Server completely handles the base64 password decoding and verification internally.

If HTTP_AUTH_RAW is set to 1, the F-Server does not need a base64 decoder. Instead of decoding and verifying both strings, it requests the application to do a string compare on the whole base64 encoded information. If there's a user / password name combination which needs to be tested, the F-Server posts this (as a zero-terminated string) via the notify code NC_HTTP_AUTH to the application-defined handler procedure and (after that) examines the return value. The application must return TRUE_stx if the credentials are correct - otherwise FALSE_stx.

Steps to enable HTTP_AUTH_RAW

In order to enable raw base64 authentication, you have to do the following steps before building your project:

- In 'httpserv.h' / 'features.h' set HTTP_AUTH_MAX_B64_LEN to a value greater than 0. This sets the maximum length of user name/password which the F-Server accepts.
- Also in 'httpserv.h' / 'features.h' set HTTP_AUTH_RAW to 1. This will instruct the F-Server to use raw base64 authentication
- Implement a branch in the callback procedure to handle NC_HTTP_AUTH events. At this place, the callback procedure receives a pointer (in the 'long' parameter).
- Get a base64 encoded representation of the user name/password combo which must be exactly the same string, as a web browser includes into its HTTP request. For example: If your name is 'Johnny' and the password is 'Rotten', the first to do is a concatenation with a colon between them: 'Johnny:Rotten'. Next you must

convert this into base64. For that purpose, you may use the little free base64 utility from our website. There are also some online tools around the internet e.g. at http://www.opinionatedgeek.com/dotnet/tools/Base64Encode/

- Implement a simple string compare using the base64 decoded string and the long value which is posted to the handle procedure when a NC_HTTP_AUTH event arrives. Return TRUE_stx on matching strings, FALSE_stx on different.

On the other hand, the F-Server has a more user-friendly interface to handle base64 authentication. It is activated when HTTP_AUTH_RAW is set to 0.

In order to enable authentication with automatic base64 decoding, you have to do the following steps before building your project:

- In 'httpserv.h' / 'features.h' define (or uncomment) HTTP_AUTH_MAX_B64_LEN. This also sets the maximum length of user name/password which the F-Server accepts.
- Be sure to set HTTP_AUTH_RAW to 0.
- Include the file 'b64dec.c' into your project.

Manage Users and Group rights

Here you can configure the access rights to the webserver pages on user group level.

Per default RessourceGen.exe generates code so that every group can access every webserver page without authentication. You can see it in the generated function stxWebsite_rsc_Init() in website_rsc.c where the last entry is always 0x00. This entry is a bit mask whereas every bit represents one user group. Altogether eight groups are supported. You might use the predefined group defines HTTP_AUTH_GRx, defined in 'httpserv.h'.

One or more groups can be assigned dynamically to a user. Before that you have to generate some users with the function **stxHTTPServ_AuthAddUser()**. HTTP_AUTH_COUNT_USER defines the maximum count of users you can add. The function has the username, password and groups flags as parameter. It returns the user index, which must be saved for later user right manipulations. If the function fails (e.g. the maximum count of users has reached) the function returns UINT8_stx_MAX.

To dynamically change user rights, please use the user index and call one of the following functions, which best fulfil your requirements: **stxHTTPServ_AuthAssignGroupToUser()** to simply add one group (group parameter as decimal value 0..8) to a user, or use **stxHTTPServ_AuthSetUserRights()** to control/clear all 8 groups of a user with a single call (group parameter as a bit masked value). To get groups rights from a user call **stxHTTPServ_AuthGetUserRights()**.

For dynamic pagenames (that are not defined in stxWebsite_rsc_Init() and which content is requested from user via NC_HTTP_USERFILE) the group rights are defined in HTTP_AUTH_GROUPS_USERFILE (group bit mask; 0xFF per default). Exception: If the pagename is “favicon.ico”, the group mask is 0x00 (-> Favicons are available for everyone without authentication).

If everything went right, a web browser should pop up a dialogue box asking for user name and password, when it connects to the F-Server. If the credentials a correct and if the user belongs to a group that is to be allowed to see the requested page, the page is sent to the browser. In the other case the dialogue box is pops up again.

14 File upload using HTML

Uploading a file using HTML does not need specials like JAVA script or cookies. You have to build a HTML FORM of the encoding type “multipart/form-data”. The FORM have to contain an INPUT tag of type “file” and of course a submit button (a INPUT tag of type “submit”).

File upload using HTML

```
<form method="post" enctype="multipart/form-data">
<INPUT name="upload_filename" type="file" size="20" accept="text/*"><br>
<INPUT type="submit" name="upload" value="Upload">
</form>
```

The file INPUT type tag:

This tag represents a file select INPUT element. Most browsers will present it as a text box with a “select” button. The file name can be entered at the text box or be selected at a special file dialogue by pressing the “select” button.

To identify the file, you should use a unique name for each **name** attribute. E.g., if you want to upload a special configuration file from the browser to your embedded device, you might use name=”config_file”. If the F-Server generates a NC_HTTP_POSTINFO_FILE event, it delivers this value at HTTP_POSTINFO.name at the HTTP_POSTINFO structure. The name of the selected file will be stored at HTTP_POSTINFO.value. Be aware to set HTTP_MAX_CTRLVALUE_LEN to a value greater than the expected file name! Otherwise a HTTPERR_CTRLVALUE_OVERFLOW is generated and the file name will be sent to application in shorter (HTTP_MAX_CTRLVALUE_LEN) length.

The **size** attribute defines the max. number of displayable characters of the file name. This enables you to define the width of the file names field length in the browser dialog box.

If you want to limit the maximum size of a file to upload, add the **maxlength** attribute. Setting this attribute to 1024 for example limits uploading files to 1024 bytes each. But be aware that most browsers do not support this feature so far! So you have to be prepared for any file size (up to 4GB) at embedded side.

Most web browsers require a additional **accept** attribute at the file INPUT tag which specifies the type of file to upload. You can set this value to “text/*”.

The submit INPUT type tag:

This is a standard submit button. There are no special attributes needed. The F-Server generates no NC_HTTP_POSTINFO event for this element or any other than the “file INPUT type” tag!

Please Note: If you want to use the file upload via HTML, you have to #define HTTP_FILEUPLOAD_ENABLED!

The information furnished in this document is believed to be accurate and reliable. However, no responsibility is assumed by sevenstax for its use, nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of sevenstax.

This document is an intellectual property of sevenstax GmbH. Unauthorized copying and distribution is prohibited.

Copyright (c) 2006 by sevenstax GmbH